

PATENT

UNITED STATES PATENT AND TRADEMARK OFFICE

Appl. No. : 10/037,390
Applicant : Wilkinson et al.
Filed : 10/23/2001
TC/A.U. : 2124
Examiner : Chavis, John Q.

Docket No. : 40.0010 C1
Customer No. : 26751

Confirmation No. 7729

#15
f Cotton
1-13-04

Certificate of Mailing

I hereby certify under 37 CFR 1.8 that this correspondence is being deposited with the United States Postal Service with sufficient postage as first class mail in an envelope addressed to: Mail Stop Appeal Brief-Patent, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on December 23, 2003.

Typed or printed name of person signing this certificate:

Mary L. Thompson

Signature: Mary L. Thompson

Mail Stop Appeal Brief-Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

APPEAL BRIEF

1. Real Party in Interest.

The real party in interest is Schlumberger Technologies, Inc., the assignee of the patent application.

2. Related appeals and interferences.

This application is not subject to any other appeals or interferences.

3. Status of Claims.

Claims 106-149 are pending in the application. Claims 106-118, 120-128, 130-143, 145, 147, and 149 stand rejected. Claims 119, 129, 144, 146 and 148 stand objected to as dependent on rejected base claims.

1/11/04
#13

4. Status of Amendments

An amendment to the specification is filed herewith. This amendment merely addresses the Examiner's objection to the specification that the appendix should be submitted on CD-ROM as it is longer than 10 pages in length. The amendment does not add any new matter or any new issues for this appeal. Entry of the amendment prior to appeal is requested.

5. Summary of the Invention

The invention is a microcontroller, an integrated circuit card, and a method having in common making it possible to develop applications (Fig. 2, Ref. No. 20) for the microcontroller or integrated circuit card (Fig. 2, Ref. No. 10) using a high level language and standard programming tools (Fig. 2, Ref. No. 22). The invention includes a converter (Fig. 2, Ref. No. 26) that accepts as input the output (Fig. 2, Ref. No. 24) from the standard programming tools. The converter converts the output – or compiled form – into programs derived from the output of the standard programming tools into a form suitable for being interpreted by an interpreter (Fig. 1, Ref. No. 16) configured to interpret programs in the converted form. In one embodiment of the invention, the converter (Fig. 3, Ref. No. 26) converts strings to identifiers.

6. Issues

Whether claims 106-118, 120-128, 130-143, 145, 147, and 149 are unpatentable over Peyret (US. Pat. No. 5,923,884) in view of Renner (U.S. Pat. No. 5,679,945) under 35 USC 103(a).

7. Grouping of the Claims

Group I: Claims 106-114, 118, 120-126.

Group II: Claims 115-117, 127-128, 133-142, 143, 145, 147, 149

Group III: Claims 119, 129, 144, 146, 148

Group I claims stand and fall together. Group II claims stand and fall together.
Group III claims stand and fall together.

Argument

Grouping of the claims. Group I: Claim 106 is representative of this group. Claim 106 recites the limitations of "a derivative application ...derived [by] converting the compiled form [of the application] into a converted form" and "an interpreter configured to interpret derivative applications in the converted form."

Group II: The claims in Group II, directly or indirectly, contain similar limitations to the Group I claims related to a converter for converting compiled form programs and an interpreter suitable for interpreting such converted form of the programs. Thus, Group II claims stand with Group I claims. However, Group II claims recited further limitations by the virtue of which they do not fall with the Group I claims. Notably, Group II claims recite limitations dealing with the conversion of strings to identifiers. Claim 143 is representative of this group. It recites "the converting step including modifying byte code operands from references using identifying strings to references using unique identifiers."

Group III: Group III claims recite that the converting step comprises at least one step of the steps of "recording all jumps and their destinations in the original byte codes; converting specific byte codes into equivalent generic byte codes or vice-versa; modifying byte code operands from references using identifying strings to references using unique identifiers; and renumbering byte codes in the compiled form to equivalent byte codes in the format suitable for interpretation" (Claim 119, representative of the group). Note: Group III claims stand allowable if rewritten in independent form.

8. Argument:

Issue: 35 USC 103(a)

Claims 106-118, 120-128, 130-143, 145, 147 and 149 were rejected under 35 U.S.C. 103(a) as being unpatentable over Peyret et al. (U.S. Patent Number 5,923,884) in

view of Renner et al. (5,679,945). Appellants traverse the rejection and respectfully request its reversal and allowance of the claims.

Group I: The Examiner has failed to establish a *prima facie* case of obviousness. "To establish a *prima facie* case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach or suggest all the claim limitations." MPEP 2143. The Examiner has failed to meet this burden.

Several limitations of the Group I claims add patentable weight thereto and Appellants do not concede that Peyret teaches or suggests any one limitation recited by these claims. That said, Appellants invite the Board to consider in particular one distinguishing feature, namely, the converter that converts the compiled form of an application into a form suitable for interpretation by a specialized interpreter that interprets derivative applications in the converted form. Claim 106 recites "first compiling the application having a class file format into a compiled form and then converting the compiled form into a converted form" and "an interpreter configured to interpret derivative applications in the converted form and derived from applications having a class file format". No such limitations are taught or suggested by Peyret or Renner taken singly or in combination.

Taking Java as an example, the prior art programming and execution sequence of a program includes three steps: writing a program in the high level Java language, compiling the program into a compiled form, and interpreting the compiled form on a Java Virtual Machine. Appellants concede these steps to be in the prior art, for example, as stated in the statement from Peyret quoted in the Office Action ("the applets run through the interpreter so that the applets do not have any direct access to the hardware of the smart card", Peyret, Col. 5, lines 44-46, quoted at Office Action, Page 4) and as taught by Rodley (cited in the Office Action only to indicate inherent features of the Java programming language).

Appellants recognized the difficulty in operating Java (or other high level language) programs within the limited resources of an integrated circuit card or other microcontroller. To solve that problem, Appellants introduced the additional step of converting the compiled form from a Java compiler into a form suitable for interpretation on a specialized interpreter. Peyret does not teach or suggest this additional step.

The Examiner has made the erroneous assertion that "converting of the compiled application ... appears to have been obvious to a person of ordinary skill in the art at the time of the invention in view of Peyret alone." Office Action, page 3, right hand column, lines 12 – 15. There is absolutely no hint in Peyret of doing a conversion step. On the contrary, Peyret states, "To execute an applet on an interpreter, as shown, source code 46 of an applet is compiled into a byte code 48. The byte code may then be executed by an interpreter on any smart card." From this foregoing quote from Peyret it must be clear that Peyret contemplated that his byte codes would go directly from compilation to interpretation.

The Examiner has further relied on Renner to teach the conversion step:

"Which infers (sic, "implies"?) that they may not all have a common structure and therefore require conversions. However, the feature is considered to be taught by Renner in an analogous art to enable to enable (sic) software with different functions and interfacing to communicate, see fig. 4 and Renner's claim 1" (Office Action, Page 3, lines 18 – 25).

The conversion step in Appellants' invention is not to address "that they may not all have a common structure".

To combine two references there must be some motivation to combine the respective teachings. What the Examiner is trying to arrive at is the sequence compile-convert-interpret, where Peyret has not taught *convert*. To resolve the issue as to whether one would be motivated to combine a "conversion" step from Renner with the compile-interpret sequence of Peyret requires that one examines what it is that Renner is proposing to convert and what is the end result of that conversion; an analysis that shows that Renner's "conversion" has absolutely nothing in common with Appellants' converter.

Renner teaches a system in which smart cards of various types may be used to replace magnetic stripe readers, bar code readers, and Wiegand effect readers. The "intelligent card reader [of Renner] can replace the aforementioned readers and yet remain compatible with their existing interface by emulating a magnetic card reader, a Wiegand effect reader, or a bar code reader." Renner, Abstract. To achieve this result, Renner teaches "conversion" from the smart card format to target device format. For example, Renner teaches "The intelligent card reader reads the preprogrammed code from the smart card, **converts** the code into Wiegand effect signals, and transmits the Wiegand effect signals over wires to an external device which normally expects to receive such signals" (Renner, Col. 6, lines 6-10, emphasis added) and "Finally, emulation functions 402 include those emulation features needed to **convert** a given data item to a magnetic stripe signal, Wiegand effect signal, or a bar code compatible signal as described above" (Renner, Col. 9, lines 40-44, emphasis added). As every person skilled in the art would know, the data that is likely to be stored on a mag stripe, for a Wiegand effect signal, or bar code would not be equivalent to a compiled application program. There is no disclosure in Renner that teaches or suggests any other type of conversion and certainly no disclosure of anything remotely resembling "converting the compiled form into a converted form" and "an interpreter configured to interpret derivative applications in the converted form" (Claim 106). How could someone who is trying to execute high level programs on a smart card be motivated to combine a compile-interpret sequence with a step of converting "a given data item to a magnetic stripe signal, Wiegand effect signal, or a bar code compatible signal" (Renner)? Thus, a person trying to solve a problem of how to take the output of a high level language compiler and interpret that code on a smart card would not likely turn to the conversion in Renner for that solution. Thus, there is no motivation in Renner or Peyret to combine their respective teachings.

Furthermore, the prior art references when combined must teach or suggest all the claim limitations. As discussed above, the conversion step in Renner solves a totally different problem. Nothing in Renner suggest conversion of a compiled form or

interpreting derivative applications in the converted form. Just because Renner uses the word *conversion* does not meet the claim of "converting the compiled form into a converted form" and "an interpreter configured to interpret derivative applications in the converted form" (Claim 106).

For the foregoing reasons, Claim 106 is not obvious over Peyret in view of Renner taken singly or in combination.

Returning now to the Examiner's statement that "in reference to the converting of the compiled application, the feature appears to have been obvious to a person of ordinary skill in the art at the time of the invention in view of Peyret alone". The Examiner has not provided any evidence (beyond Renner which Appellants have demonstrated to be immaterial) to support that point. Appellants posit, using the argument follows, that the opposite is true (namely that the claimed method is not obvious).

It is well-established law that to render a claim obvious, a prior art reference must provide an enabling disclosure. *Rockwell Int'l v United States*, 47 USPQ2d 1027, 1032 (CAFC 1998), *Motorola, Inc. v. Interdigital Technology Corp.* 43 USPQ2d 1481, 1489 (CAFC 1997), *Beckman Instruments, Inc. v. LKB Produkter AB*, 13 USPQ2d 1301, 1304 (CAFC 1989). Peyret is completely silent on how to enable a program written in a high level language to operate on an integrated circuit card.

To put Java (or any other high level language) on an integrated circuit card is anything but obvious. At the time of the invention, the typical Java Virtual Machine required over 1 MB of memory. Any person of ordinary skill would realize that to squeeze such an interpreter into an integrated circuit card (such as a smart card) is anything but an obvious task. Appellants direct the Board to the following statement made about the author of a recent JAVA WORLD article (<http://www.javaworld.com/javaworld/jw-04-1998/jw-04-ringfever.html>):

“About the author: Chuck McManis wrote the very first Crypto toolkit in Java when he was a member of the Java development group in 1993. **He also brought in the first smart cards (Hitachi) at FirstPerson but gave up trying to put Java on them.** (Now he knows better!) Today he writes columns and articles for JavaWorld and is currently holding the position of director of systems software at FreeGate Corporation in Sunnyvale.”

Appellants submit that Chuck McManis is well beyond a person of ordinary skill in the art of Java programming. In fact, he is a leading expert in the field of software design and holds at least four patents for his work. The fact that he “gave up trying to put Java on [smart cards]” is indicative of the non-obviousness of doing so.

Appellants also invite the Board to consider the following excerpt including a quote from Scott McNealy (CEO of Sun Microsystems, the company that originally developed the Java language):

"Like Playing Golf in a Phone Booth"

A smart card enabled by Java Card technology, dubbed the "world's thinnest computer," places the Java platform inside the .8 millimeter thickness of the plastic card, thus making it a small but integral part of a larger enterprise information system.

Its creation was no small feat. "Fitting Java technology inside smart cards was like playing golf inside a telephone booth," remarks Sun CEO Scott McNealy.

<http://softwarema.ussec.sun.com/features/1999/01/javacard.html>

JAVA CARDTM TECHNOLOGY GROWS UP SMART

by Janice J. Heiss and John Papageorge

When two *experts* (McMannis and McNealy) in the field of the Java programming language consider putting Java on smart cards very difficult, for a person of *ordinary* skill it would be nearly unattainable and certainly not obvious. Peyret's disclosure, which neither mentions nor suggests high level languages nor how to put such language interpreters on a smart card, would be of no help to the ordinarily skilled person.

The Examiner has suggested that Peyret as teaching the claimed limitations.

For example, the Examiner pointed to Peyret Fig. 1, the abstract and col. 1 lines 4-16, 59 – 67 and col. 5 lines 59-67 of Peyret (Office Action, Page 3). Appellants have not found one hint of "converting the compiled form into a converted form" in these passages of Peyret. Figure 1 of Peyret merely shows a smart card architecture having a ROM, a RAM, an NVM (non-volatile memory) all connected to a CPU, which in turn is connected to an I/O line. To infer that Peyret teaches or suggests, "converting the compiled form into a converted form" from Figure 1 is impossible.

The Abstract of Peyret states the following:

A system for loading an applet and its associated use rights into a smart card having other applets with associated use rights with values that change as the application is used is provided that stores, remotely from said smart card, an applet and use rights with a predetermined initial value, associated with the applet, and has a smart card having a processing unit, and a memory unit, the memory unit being connected to the processing unit and storing a second application having use rights. The smart card may be connected to said remote storage means, and the application, having use rights with a predetermined value, may be loaded from said remote storage means into said smart card. A smart card is also provided having a processor for executing an application, a memory, connected to the processor, for storing multiple applications, including a first application having first use rights and having first values associated with the first use rights, the first value changing from a predetermined initial value with use of the first use rights, a system for loading in the smart card a second application from a remote location over an interface, the second application having second use rights, a system for storing said second application into said memory in said smart card, and a system for changing the use rights of said first application and said second application. A method of replenishing the use rights in a smart card is also provided.

Thus, the Abstract deals with the loading of applets and associated use rights, the architecture of the smart card. It concerns the use rights of the smart card, the changing

of the use rights and of loading a second application onto the smart card. There is absolutely no hint of converting compiled applets into another form.

Col. 1, lines 4 – 16 describes a smart card as being a plastic, credit card-sized card having a semiconductor chip for executing simple programs. The passage further describes some of the applications of smart cards. The passage does not describe how these programs are created. Therefore, it is not surprising that there is no hint of converting compiled programs into another form.

Col. 1, lines 59-67 states that "permanent smart cards have use rights that may be replenished", gives some examples of permanent smart cards, and states that "these permanent smart cards have more memory for multiple applets and the use rights on the smart card may be separately and independently replenished." Again, there is no suggestion of how applets are created and, therefore not surprisingly, no suggestion of converting a compiled form into a converted form.

Col. 5, lines 59-67 is interesting because it teaches away from Appellants' invention. The passage states, "to execute an applet on an interpreter, as shown, source code 46 of an applet is compiled into a byte code 48. The byte code may then be executed by any interpreter on any smart card." Thus, Peyret teaches that the compiled form of the applet is executed by the smart card interpreter. In contrast to that teaching, Appellants have introduced "converting the compiled form into a converted form" and "an interpreter configured to interpret derivative applications in the converted form". Peyret does not teach these limitations. It is improper to infer such a teaching from the disclosure of Peyret, generally, and in particular, from the passages relied upon by the Examiner.

In rejecting Claims 112-113, the Examiner makes the statement that "the converter being inherent via col. 1 lines 33-52 and col. 5 lines 48-58." The Examiner has not met the legal standard for finding that a limitation is inherent in a reference. It is well

established law that "if the prior art reference does not expressly set forth a particular element of the claim, that reference still may anticipate if that element is 'inherent' in its disclosure. To establish inherency, the extrinsic evidence 'must make clear that the missing descriptive matter is necessarily present in the thing described in the reference, and that it would be so recognized by persons of ordinary skill.'" *In re Robertson and Scripps*, 169 F.3d 743, 49 USPQ2d, 1949, 1950-1951 (Fed. Cir. 1999) *citing*, *Continental Can Co. v. Monsanto Co.*, 948 F.2d 1264, 1268, 20 U.S.P.Q.2d 1746, 1749 (Fed. Cir. 1991). The Examiner has failed to point to any extrinsic evidence that would make clear that the converter is necessarily present in the system described in Peyret and that the converter would be so recognized by a person of ordinary skill in the art. Accordingly, Appellants respectfully request that the Board overturns any rejection that the converter is inherent in Peyret as that contention does not meet the legal standard set forth by the Federal Circuit in *Continental Can* and, further, in *In re Robertson and Scripps*.

For these reasons, Appellants respectfully submit that the Examiner has not established a *prima facie* case of obviousness under 35 USC 103(a) of the Group I claims. Accordingly, Appellants request the reversal of the rejection of the Group I claims and their early allowance.

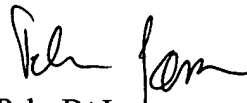
Group II: The Group II claims recite similar limitations to the Group I claims and therefore stand with the Group I claims. Furthermore, these claims recite additional limitations, for example, as set forth in Claim 143, "the converting step including modifying byte codes operands from references using identifying strings to references using unique identifiers." As noted above in support of the Group I claims, Peyret and Renner do not teach or suggest expressly or inherently a converter as claimed either singly or in combination. It is therefore not surprising that these references also fail to teach or suggest this novel and non-obvious conversion step. For this reason Appellants respectfully request reversal of the rejection of the Group II claims and their early allowance.

Group III: The Group III claims stand allowed if rewritten in independent form. However, in light of the arguments herein in support of the Group I and II claims, Appellants decline to so rewrite the Group III claims.

It is submitted that all of the claims now before the Board of Appeals are allowable. Appellant respectfully requests reversal of the rejections set forth by the Examiner and early allowance of the Application.

It is believed that except for the fee for submitting an appeal brief under 37 CFR 1.17(c), which is submitted herewith, no additional fees or other fees are due in connection with this Response as has been indicated on the fee transmittal sheet. If Appellant is in error as to these fees, the Commissioner is hereby authorized to charge any fees that may be required, or credit any overpayment, to Deposit Account 19-0597.

Respectfully submitted,



Pehr B. Jansson
Registration No. 35,759

Date: December 23, 2003

Enclosures:

1. Return Receipt Postcard
2. This Appeal Brief (13 pages) and Appendix (11 pages) in triplicate
3. Transmittal Form (1 page)
4. Amendment Under CFR 1.116 (4 pages)
5. Appendices A-I on CD-ROM
6. Notice of Appeal (1 page)
7. Fee Transmittal for FY 2004 w/Authorization to Charge Deposit Account (1 page)

Customer No. 26751
Schlumberger Austin Technology Center
Attn: Pehr B. Jansson, Intellectual Property Law Dept.
8311 North FM 620
Austin, TX 78726
Tel: 512-331-3748
Fax: 512-331-3060

9. Appendix: Currently Pending Claims

106. A microcontroller comprising:

a memory storing:

a derivative application derived from an application having a class file format wherein the application is derived from an application having a class file format by first compiling the application having a class file format into a compiled form and then converting the compiled form into a converted form, and

an interpreter configured to interpret derivative applications in the converted form and derived from applications having a class file format; and

a processor coupled to the memory, the processor configured to use the interpreter to interpret the derivative application for execution.

107. The microcontroller of claim 106, further comprising:

a communicator configured to communicate with a terminal.

108. The microcontroller of claim 107, wherein the terminal has a card reader and the communicator comprises a contact for communicating with the card reader.

109. The microcontroller of claim 108, wherein the terminal has a wireless communicator and a wireless transceiver for communicating with the wireless communication device.

110. The microcontroller of claim 108, wherein the terminal has a wireless communication device and the communicator comprises a wireless transmitter for communicating with the wireless communication device.

111. The microcontroller of claim 106, wherein the class file format comprises a Java class file format.

112. A microcontroller having a set of resource constraints and comprising:

a memory, and

an interpreter loaded in memory and operable within the set of resource constraints, the microcontroller having: at least one application loaded in the memory to be interpreted by the interpreter, wherein the at least one application is generated by a programming environment comprising:

- a) a compiler for compiling application source programs written in high level language source code form into a compiled form, and
- b) a converter for post processing the compiled form into a minimized form suitable for interpretation within the set of resource constraints by the interpreter.

113. The microcontroller of Claim 112, wherein the compiled form includes attributes, and the converter comprises a means for including attributes required by the interpreter while not including the attributes not required by the interpreter.

114. The microcontroller of Claim 112 wherein the compiled form is in a standard Java class file format and the converter accepts as input the compiled form in the standard Java class file format and produces output in a form suitable for interpretation by the interpreter.

115. The microcontroller of Claim 112 wherein the compiled form includes associating an identifying string for objects, classes, fields, or methods, and the converter comprises a means for mapping such strings to unique identifiers.

116. The microcontroller of Claim 115 wherein each unique identifier is an integer.

117. The microcontroller of Claim 115 wherein the mapping of strings to unique identifiers is stored in a string to identifier map file.

118. The microcontroller of Claim 112 where in the high level language supports a first set of features and a first set of data types and the interpreter supports a subset of the first set of features and a subset of the first set of data types, and wherein the converter verifies that the compiled form only contains features in the subset of the first set of features and only contains data types in the subset of the first set of data types.

119. The microcontroller of Claim 115 wherein the compiled form is in a byte code format and the converter comprises means for translating from the byte codes in the compiled form to byte codes in a format suitable for interpretation by the interpreter by:

using at least one step in a process including the steps:

- a) recording all jumps and their destinations in the original byte codes;
- b) converting specific byte codes into equivalent generic byte codes or vice-versa;
- c) modifying byte code operands from references using identifying strings to references using unique identifiers; and
- d) renumbering byte codes in the compiled form to equivalent byte codes in the format suitable for interpretation; and

relinking jumps for which destination address is effected by conversion step a), b), c), or d).

120. The microcontroller of Claim 112 wherein the application program is compiled into a compiled form for which resources required to execute or interpret the compiled form exceed those available on the microcontroller.

121. The microcontroller of Claim 112 wherein the compiled form is designed for portability on different computer platforms.

122. The microcontroller of Claim 112 wherein the interpreter is further configured to determine, during an interpretation of an application, whether the application meets a security criteria selected from a set of rules containing at least one rule selected from the set:

not allowing the application access to unauthorized portions of memory,
not allowing the application access to unauthorized microcontroller resources,
wherein the application is composed of byte codes and checking a plurality of byte codes at least once prior to execution to verify that execution of the byte codes does not violate a security constraint.

123. The microcontroller of Claim 112 wherein at least one application program is generated by a process including the steps of:

prior to loading the application verifying that the application does not violate any security constraints; and
loading the application in a secure manner.

124. The microcontroller of Claim 123 wherein the step of loading in a secure manner comprises the step of:

verifying that the loading identity has permission to load applications onto the microcontroller.

125. The microcontroller of Claim 123 wherein the step of loading in a secure manner comprises the step of:

encrypting the application to be loaded using a loading key.

126. A method of programming a microcontroller having a memory and a processor operating according to a set of resource constraints, the method comprising the steps of:

inputting an application program in a first programming language;

compiling the application program in the first programming language into a first intermediate code associated with the first programming language, wherein the first intermediate code being interpretable by at least one first intermediate code virtual machine;

converting the first intermediate code into a second intermediate code; wherein the second intermediate code is interpretable within the set of resource constraints by at least one second intermediate code virtual machine; and

loading the second intermediate code into the memory of the microcontroller.

127. The method of programming a microcontroller of Claim 126 wherein the step of converting further comprises:

associating an identifying string for objects, classes, fields, or methods; and mapping such strings to unique identifiers.

128. The method of Claim 127 wherein the step of mapping comprises the step of mapping strings to integers.

129. The method of Claim 126 wherein the step of converting comprises at least one of the steps of:

- a) recording all jumps and their destinations in the original byte codes;
- b) converting specific byte codes into equivalent generic byte codes or vice-versa;
- c) modifying byte code operands from references using identifying strings to references using unique identifiers;
- d) renumbering byte codes in a compiled format to equivalent byte codes in a format suitable for interpretation; and
- e) relinking jumps for which destination address is effected by conversion step a), b), c), or d).

130. The method of Claim 126 wherein the step of loading the second intermediate code into the memory of the microcontroller further comprises checking the second intermediate code prior to loading the second intermediate code to verify that the second intermediate code meets a predefined integrity check and that loading is performed according to a security protocol.

131. The method of Claim 130 wherein the security protocol requires that a particular identity must be validated to permit loading prior to the loading of the second intermediate code.

132. The method of Claim 130 further characterized by providing a decryption key and wherein the security protocol requires that the second intermediate code is encrypted using a loading key corresponding to the decryption key.

133. A microcontroller operable to execute derivative programs which are derivatives of programs written in an interpretable programming language having a memory and an interpreter, the microcontroller comprising:

- (a) the microcontroller operating within a set of resource constraints including the memory being of insufficient size to permit interpretation of programs written in the interpretable programming language; and
- (b) the memory containing an interpreter operable to interpret the derivative programs written in the derivative of the interpretable language wherein a derivative of a program written in the interpretable programming language is derived from the compiled version of a program written in the interpretable programming language by applying a conversion of the compiled version including applying at least one rule selected from a set of rules including:
 - (1) mapping strings to identifiers;
 - (2) performing security checks prior to or during interpretation;
 - (3) performing structural checks prior to or during interpretation; and
 - (4) performing semantic checks prior to or during interpretation.

134. The microcontroller of Claim 133 wherein the derivative programs are class files or derivatives of class files.

135. The microcontroller of Claim 133 further comprising:
the memory containing less than 1 megabyte of storage.

136. The microcontroller of Claim 133 wherein the security checks the microcontroller is further comprising:

- (c) logic to receive a request from a requester to access one of a plurality of derivative programs;
- (d) after receipt of the request, determine whether the one of a plurality of derivative programs complies with a predetermined set of rules; and
- (e) based on the determination, selectively grant access to the requester to the one of the plurality of applications.

137. The microcontroller of Claim 136, wherein the predetermined rules are enforced by the interpreter while the derivative program is being interpreted by determining whether the derivative program has access rights to a particular part of memory the derivative program is attempting to access.

138. The microcontroller of Claim 133 further wherein the microcontroller is configured to perform at least one security check selected from the set having the members:

- (a) enforcing predetermined security rules while the derivative program is being interpreted, thereby preventing the derivative program from accessing unauthorized portions of memory or other unauthorized microcontroller resources,

- (b) the interpreter being configured to check each bytecode at least once prior to execution to determine that the bytecode can be executed in accordance with pre-execution and post-execution checks, and
- (c) the derivative program is checked prior to being loaded into the microcontroller to verify the integrity of the derivative program and loading is performed according to a security protocol.

139. The microcontroller of Claim 138 wherein the security protocol requires that a particular identity must be validated to permit loading a derivative program onto a card.

140. The microcontroller of Claim 138 further comprising a decryption key wherein the security protocol requires that a derivative program to be loaded is encrypted using a loading key corresponding to the decryption key.

141. The microcontroller of Claim 133 wherein the microcontroller is configured to provide cryptographic services selected from the set including encryption, decryption, signing, signature verification, mutual authentication, transport keys, and session keys.

142. The microcontroller of Claim 133 further comprising a file system and wherein the microcontroller is configured to provide secure access to the file system through a means selected from the set including:

- (a) the microcontroller having access control lists for authorizing reading from a file, writing to a file, or deletion of a file,
- (b) the microcontroller enforcing key validation to establish the authorized access to a file, and
- (c) the microcontroller verifying card holder identity to establish the authorized access to a file.

143. An integrated circuit card for use with a terminal, comprising:

a communicator configured to communicate with the terminal;

a memory storing:

an application derived from a program written in a high level programming language format wherein the application is derived from a program written in a high level programming language format by first compiling the program into a compiled form and then converting the compiled form into a converted form, the converting step including modifying byte code operands from references using identifying strings to references using unique identifiers; and

an interpreter operable to interpret such a derivative application in the converted form and derived from a program written in a high level programming language format; and

a processor coupled to the memory, the processor configured to use the interpreter to interpret the application for execution and to use the communicator to communicate with the terminal.

144. The integrated circuit card of Claim 143 wherein the converting step further comprises:

recording all jumps and their destinations in the original byte codes;
converting specific byte codes into equivalent generic byte codes or vice-versa;
and

renumbering byte codes in a compiled format to equivalent byte codes in a format suitable for interpretation.

145. A method for use with an integrated circuit card and a terminal, comprising:

storing an interpreter operable to interpret programs derived from programs written in a high level programming language and an application derived from a program written in a high level programming language format in a memory of the integrated circuit card wherein the application is derived from a program written in a high level programming language format by first compiling the program into a compiled form and

then converting the compiled form into a converted form, the converting step including modifying byte code operands from references using identifying strings to references using unique identifiers; and

using a processor of the integrated circuit card to use the interpreter to interpret the application for execution; and

using a communicator of the card when communicating between the processor and the terminal.

146. The method of Claim 145 wherein the converting step further comprises:
recording all jumps and their destinations in the original byte codes;
converting specific byte codes into equivalent generic byte codes or vice-versa;
and

renumbering byte codes in a compiled format to equivalent byte codes in a format suitable for interpretation.

147. An integrated circuit card for use with a terminal, comprising:
a communicator configured to communicate with the terminal;
a memory storing:
applications, each application derived from applications having a high level programming language format, and
an interpreter operable to interpret applications derived from applications having a high level programming language format wherein the application is derived from a program written in a high level programming language format by first compiling the program into a compiled form and then converting the compiled form into a converted form, the converting step including modifying byte code operands from references using identifying strings to references using unique identifiers; and

a processor coupled to the memory, the processor configured to:

- a.) use the interpreter to interpret the applications for execution,
- b.) use the interpreter to create a firewall to isolate the applications from each other, and
- c.) use the communicator to communicate with the terminal.

148. The integrated circuit card of Claim 147 wherein the interpreter is further operable to interpret applications derived using a converting step including:

recording all jumps and their destinations in the original byte codes;

converting specific byte codes into equivalent generic byte codes or vice-versa;

and

renumbering byte codes in a compiled format to equivalent byte codes in a format suitable for interpretation.

149. A microcontroller operable to execute derivative programs which are derivatives of programs written in an interpretable programming language having a memory and an interpreter, the microcontroller comprising:

the microcontroller operating within a set of resource constraints including the memory being of insufficient size to permit interpretation of programs written in the interpretable programming language; and

the memory containing an interpreter operable to interpret the derivative programs written in the derivative of the interpretable language wherein a derivative of a program written in the interpretable programming language is derived from a compiled for of the program written in the interpretable programming language by performing a conversion including mapping strings to identifiers.